Underrated Machine Learning Algorithms — APRIORI

Building step by step module from scratch using Python

Harsha Dannina Jan 28, 2020 · 7 min read ★

Introduction to APRIORI

Apriori is an algorithm used for Association Rule Mining. It searches for a series of frequent sets of items in the datasets. It builds on associations and correlations between the itemsets. It is the algorithm behind "You may also like" where you commonly saw in recommendation platforms.

What is Associate Rule Mining?

ARM(Associate Rule Mining) is one of the important techniques in data science. In ARM, the frequency of patterns and associations in the dataset is identified among the item sets then used to predict the next relevant item in the set. This ARM technique is mostly used in business decisions according to customer purchases.

Example: In Walmart, if Ashok buys Milk and Bread, the chances of him buying Butter are predicted by the Associate Rule Mining technique.

> *In this module, We are building an apriori algorithm from scratch on the given dataset and used it as a recommendation system for customer purchases.*

Some definitions need to be remembered

Before we start, go through some terms which are explained below.

SUPPORT_COUNT — number of transactions in which the itemset appears.

MINIMUM_SUPPORT_COUNT — the minimum frequency of itemset in the dataset.

CANDIDATE_SET — C(k) support_count of each item in the dataset.

ITEM_SET — L(k) comparing each item in the candidate_set support count to minimum_support_count and filtering the under frequent itemset.

SUPPORT — the percentage of transactions in the database follow the rule.

```
Support(A->B) = Support_count(A U B)
```

CONFIDENCE — the percentage of customers who bought A also bought B.

```
Confidence(A->B) = [Support_count(AUB)/Support_count(A)]*100
```

Get the Data

For this experiment, We considered a dataset called Grocery Store Data set from Kaggle. It consists of 20 transactions of general items from a supermarket. This dataset is easier to understand the patterns and associations.

MILK	BREAD	BISCUIT	
BREAD	MILK	BISCUIT	CORNFLAKES
BREAD	TEA	BOURNVITA	
JAM	MAGGI	BREAD	MILK
MAGGI	TEA	BISCUIT	
BREAD	TEA	BOURNVITA	
MAGGI	TEA	CORNFLAKES	
MAGGI	BREAD	TEA	BISCUIT
JAM	MAGGI	BREAD	TEA
BREAD	MILK		
COFFEE	COCK	BISCUIT	CORNFLAKES
COFFEE	COCK	BISCUIT	CORNFLAKES
COFFEE	SUGER	BOURNVITA	
BREAD	COFFEE	COCK	
BREAD	SUGER	BISCUIT	
COFFEE	SUGER	CORNFLAKES	
BREAD	SUGER	BOURNVITA	
BREAD	COFFEE	SUGER	
BREAD	COFFEE	SUGER	
TEA	MILK	COFFEE	CORNFLAKES

Grocery Store Data set

Here we can see 20 sets of transactions on grocery items i.e., MILK, BREAD, BISCUIT, CORNFLAKES, TEA, BOURNVITA, JAM, MAGGI, COFFEE, COCK, and SUGAR.

> *Though this dataset is small, we don't need to generate an argument dataset. This is sufficient to develop the Apriori algorithm.*

Prepare the data

This data need to be processed to generate records and item-list. Consider minimum_support_count to be 2. Pandas library is used to import the CSV file.

> *Prerequests: PYTHON Intermediate level*
>
> *Geeksforgeeks: Apriori Algorithm(theory-based)*

```
import pandas as pd
import itertools

data = pd.read_csv('GroceryStoreDataSet.csv')

minimum_support_count = 2
records = []
for i in range(0, 20):
    records.append([str(data.values[i,j]) for j in range(0, 4)])

items = sorted([item for sublist in records for item in sublist if
item != 'nan'])
```

Build the Algorithm

Let's revise the key property of Apriori before building the algorithm,

> *All subsets of a frequent itemset must be frequent.*
> *If an itemset is infrequent, all its supersets will be infrequent.*

Step — 1

In stage1, the candidate_set C1 is generated by measuring the support_count of each item in the dataset. Item_set L1 is generated by comparing C1 support_count with minimum_support_count. Here k=1.

```
def stage_1(items, minimum_support_count):
    c1 = {i:items.count(i) for i in items}
    l1 = {}
    for key, value in c1.items():
        if value >= minimum_support_count:
           l1[key] = value

    return c1, l1

c1, l1 = stage_1(items, minimum_support_count)
```

The outcome of step-1 is Item_set L1,

BISCUIT	7
BOURNVITA	4
BREAD	13
COCK	3
COFFEE	8
CORNFLAKES	6
JAM	2
MAGGI	5
MILK	5
SUGER	6
TEA	7

Itemset L1

In this case, there are no low frequent items to minimum_support_count in candidate_set. So Candidate_set C1=Item_set L1.

Step — 2

In this stage, candidate_set C2 is generated using Item_set L1 from the previous step. Check all subsets in itemset are frequent if not, remove respective itemset from the list. Item_set L2 is generated by comparing candidate_set C2 with minimum_support_count. Here k=2.

```
def stage_2(l1, records, minimum_support_count):
    l1 = sorted(list(l1.keys()))
    L1 = list(itertools.combinations(l1, 2))
    c2 = {}
    l2 = {}
    for iter1 in L1:
        count = 0
        for iter2 in records:
            if sublist(iter1, iter2):
                count+=1
        c2[iter1] = count
    for key, value in c2.items():
        if value >= minimum_support_count:
            if check_subset_frequency(key, l1, 1):
                l2[key] = value
```

```
    return c2, l2

c2, l2 = stage_2(l1, records, minimum_support_count)
```

To check subsets of an itemset are frequent, we should pass current stage itemset, the previous stage itemset, in this case, L1, and k-1.

```
def check_subset_frequency(itemset, l, n):
    if n>1:
        subsets = list(itertools.combinations(itemset, n))
    else:
        subsets = itemset
    for iter1 in subsets:
        if not iter1 in l:
            return False
    return True
```

The outcome of Step-2 is Item_set L2,

BISCUIT	BREAD	4
	COCK	2
	COFFEE	2
	CORNFLAKES	3
	MAGGI	2
	MILK	2
	TEA	2
BOURNVITA	BREAD	3
	SUGER	2
	TEA	2
BREAD	COFFEE	3
	JAM	2
	MAGGI	3
	MILK	4
	SUGER	4
	TEA	4
COCK	COFFEE	3
	CORNFLAKES	2
COFFEE	CORNFLAKES	4

	SUGER	4
CORNFLAKES	MILK	2
	TEA	2
JAM	MAGGI	2
MAGGI	TEA	4

Itemset L2

In this case, 31 items in itemset need to be eliminated due to low frequent category i.e., below minimum_support_count.

Step — 3

In this stage, candidate_set C3 is generated using Item_set L2 from the previous step. Check all subsets in itemset are frequent if not, remove respective itemset from the list. Item_set L3 is generated by comparing candidate_set C3 with minimum_support_count. Here k=3.

```
def stage_3(l2, records, minimum_support_count):
    l2 = list(l2.keys())
    L2 = sorted(list(set([item for t in l2 for item in t])))
    L2 = list(itertools.combinations(L2, 3))
    c3 = {}
    l3 = {}
    for iter1 in L2:
        count = 0
        for iter2 in records:
            if sublist(iter1, iter2):
                count+=1
        c3[iter1] = count
    for key, value in c3.items():
        if value >= minimum_support_count:
            if check_subset_frequency(key, l2, 2):
                l3[key] = value

    return c3, l3

c3, l3 = stage_3(l2, records, minimum_support_count)
```

The outcome of Step-3 is Item_set L3,

Itemset L3

Step — 4

In this stage, candidate_set C4 is generated using Item_set L3 from the previous step. Check all subsets in itemset are frequent if not, remove respective itemset from the list. Item_set L4 is generated by comparing candidate_set C4 with minimum_support_count. Here k=4.

```
def stage_4(l3, records, minimum_support_count):
    l3 = list(l3.keys())
    L3 = sorted(list(set([item for t in l3 for item in t])))
    L3 = list(itertools.combinations(L3, 4))
    c4 = {}
    l4 = {}
    for iter1 in L3:
        count = 0
        for iter2 in records:
            if sublist(iter1, iter2):
                count+=1
        c4[iter1] = count
    for key, value in c4.items():
        if value >= minimum_support_count:
            if check_subset_frequency(key, l3, 3):
                l4[key] = value

    return c4, l4

c4, l4 = stage_4(l3, records, minimum_support_count)
```

The outcome of Step-4 is Item_set L4,

Itemset L4

We can stop here because no frequent subsets are found further.

Generate the association rule

To generate association rule for the dataset, we need to calculate the confidence of each rule.

```
''' Rule generation of itemset '''

'''
Confidence:

Confidence(A->B)=Support_count(AUB)/Support_count(A)

Confidence((COCK, COFFEE)->CORNFLAKES) = Support_count('COCK',
'COFFEE', 'CORNFLAKES')/Support_count('COCK', 'COFFEE')

'''
```

Let's consider Item_set L3 for the association rule.

```
sets = []
for iter1 in list(l3.keys()):
    subsets = list(itertools.combinations(iter1, 2))
    sets.append(subsets)
```

Implementing association rule on generated set from L3

```
def support_count(itemset, itemlist):
    return itemlist[itemset]

list_l3 = list(l3.keys())
for i in range(0, len(list_l3)):
    for iter1 in sets[i]:
        a = iter1
        b = set(list_l3[i]) - set(iter1)
        confidence = (support_count(list_l3[i],
itemlist)/support_count(iter1, itemlist))*100
        print("Confidence{}->{} = ".format(a,b), confidence)
```

Minimum confidence is assumed to be 50%

Results

Confidence('BISCUIT', 'COCK')->{'CORNFLAKES'} = 100.0
Confidence('BISCUIT', 'CORNFLAKES')->{'COCK'} = 66.66666666666666
Confidence('COCK', 'CORNFLAKES')->{'BISCUIT'} = 100.0
Confidence('BISCUIT', 'MAGGI')->{'TEA'} = 100.0
Confidence('BISCUIT', 'TEA')->{'MAGGI'} = 100.0
Confidence('MAGGI', 'TEA')->{'BISCUIT'} = 50.0
Confidence('BISCUIT', 'COFFEE')->{'CORNFLAKES'} = 100.0
Confidence('BISCUIT', 'CORNFLAKES')->{'COFFEE'} = 66.66666666666666
Confidence('COFFEE', 'CORNFLAKES')->{'BISCUIT'} = 50.0
Confidence('BISCUIT', 'BREAD')->{'MILK'} = 50.0
Confidence('BISCUIT', 'MILK')->{'BREAD'} = 100.0
Confidence('BREAD', 'MILK')->{'BISCUIT'} = 50.0
Confidence('BREAD', 'COFFEE')->{'SUGER'} = 66.66666666666666
Confidence('BREAD', 'SUGER')->{'COFFEE'} = 50.0
Confidence('COFFEE', 'SUGER')->{'BREAD'} = 50.0
Confidence('BREAD', 'MAGGI')->{'TEA'} = 66.66666666666666
Confidence('BREAD', 'TEA')->{'MAGGI'} = 50.0
Confidence('MAGGI', 'TEA')->{'BREAD'} = 50.0
Confidence('COCK', 'COFFEE')->{'CORNFLAKES'} = 66.66666666666666
Confidence('COCK', 'CORNFLAKES')->{'COFFEE'} = 100.0
Confidence('COFFEE', 'CORNFLAKES')->{'COCK'} = 50.0
Confidence('BOURNVITA', 'BREAD')->{'TEA'} = 66.66666666666666
Confidence('BOURNVITA', 'TEA')->{'BREAD'} = 100.0
Confidence('BREAD', 'TEA')->{'BOURNVITA'} = 50.0
Confidence('BISCUIT', 'COCK')->{'COFFEE'} = 100.0
Confidence('BISCUIT', 'COFFEE')->{'COCK'} = 100.0
Confidence('COCK', 'COFFEE')->{'BISCUIT'} = 66.66666666666666
Confidence('BREAD', 'JAM')->{'MAGGI'} = 100.0
Confidence('BREAD', 'MAGGI')->{'JAM'} = 66.66666666666666
Confidence('JAM', 'MAGGI')->{'BREAD'} = 100.0

Conclusion

This association rule mining technique was implemented by giants like Amazon, Netflix, Google, Flipkart, and Spotify in their recommendation platforms.

Inspiration Feed

This algorithm also used as a marketing technique for discounts on most selling product combinations.

Today, we've learned how to build the Apriori algorithm and implementing it in Association Rule Mining on a general grocery dataset from a supermarket.

The dataset and the entire code is available at my Git repository.

Thanks to Anne Bonner.

Data Science Machine Learning Data Mining Recommendations Apriori